

2

COPY

Entered

AD-A220 411

ON PAGE

READ INSTRUCTIONS
BEFORE COMPLETING FORM

12. GOVT ACCESSION NO.

3. RECIPIENT'S CATALOG NUMBER

4. TITLE (and Subtitle)

Ada Compiler Validation Summary Report: International Business Machines Corporation, The IBM Development System for the Ada Language, AIX/RT Follow-on, Version 1.1 IBM RT Follow-on (Host & Target), 891129W1.10198

5. TYPE OF REPORT & PERIOD COVERED

29 Nov. 1989 to 29 Nov. 1990

6. PERFORMING ORG. REPORT NUMBER

7. AUTHOR(s)

Wright-Patterson AFB
Dayton, OH, USA

8. CONTRACT OR GRANT NUMBER(s)

9. PERFORMING ORGANIZATION AND ADDRESS

Wright-Patterson AFB
Dayton, OH, USA

10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS

11. CONTROLLING OFFICE NAME AND ADDRESS

Ada Joint Program Office
United States Department of Defense
Washington, DC 20301-3081

12. REPORT DATE

13. NUMBER OF PAGES

14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)

Wright-Patterson AFB
Dayton, OH, USA

15. SECURITY CLASS (of this report)
UNCLASSIFIED

15a. DECLASSIFICATION/DOWNGRADING
SCHEDULE
N/A

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20 if different from Report)

UNCLASSIFIED

18. SUPPLEMENTARY NOTES

19. KEYWORDS (Continue on reverse side if necessary and identify by block number)

Ada Programming language, Ada Compiler Validation Summary Report, Ada Compiler Validation Capability, ACVC, Validation Testing, Ada Validation Office, AVO, Ada Validation Facility, AVF, ANSI/MIL-STD-1815A, Ada Joint Program Office, AJPO

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

International Business Machines Corporation Wright-Patterson AFB, The IBM Development System for the Ada Language AIX/RT follow-on, Version 1.1, IBM RT Follow-on under AIX, Version 3.1 (Host & Target), ACVC 1.10.

DD FORM 1473 EDITION OF 1 NOV 85 IS OBSOLETE
1 JAN 73 S/N 0102-LF-014-0001

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

90 04 10 139

AVF Control Number: AVF-VSR-352.0290
89-07-06-IBM

Ada COMPILER
VALIDATION SUMMARY REPORT:
Certificate Number: 891129W1.10198
International Business Machines Corporation
The IBM Development System for the Ada Language
AIX/RT Follow-on, Version 1.1
IBM RT Follow-on



Completion of On-Site Testing:
29 November 1989

Prepared By:
Ada Validation Facility
ASD/SCOL
Wright-Patterson AFB OH 45433-6503

Prepared For:
Ada Joint Program Office
United States Department of Defense
Washington DC 20301-3081

Accession For	
NTIS GRA&I	<input checked="checked" type="checkbox"/>
DTIC TAB	<input type="checkbox"/>
Unannounced	<input type="checkbox"/>
Justification	
By	
Distribution/	
Availability Codes	
Dist	Avail and/or Special
A-1	

Ada Compiler Validation Summary Report:

Compiler Name: The IBM Development System for the Ada Language
AIX/RT Follow-on, Version 1.1

Certificate Number: 891129W1.10198


Host: IBM RT Follow-on under
AIX, Version 3.1

Target: IBM RT Follow-on under
AIX, Version 3.1


Testing Completed 29 November 1989 Using ACVC 1.10

Customer Agreement Number: 89-07-06-IBM

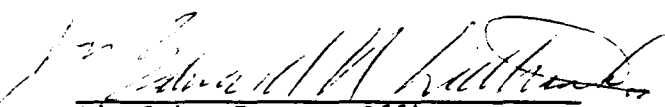
This report has been reviewed and is approved.



Ada Validation Facility
Steven P. Wilson
Technical Director
ASD/SCOL
Wright-Patterson AFB OH 45433-6503



Ada Validation Organization
fr Director, Computer & Software Engineering Division
Institute for Defense Analyses
Alexandria VA 22311



Ada Joint Program Office
Dr. John Solomond
Director
Department of Defense
Washington DC 20301

TABLE OF CONTENTS

CHAPTER 1	INTRODUCTION	
1.1	PURPOSE OF THIS VALIDATION SUMMARY REPORT	1-2
1.2	USE OF THIS VALIDATION SUMMARY REPORT	1-2
1.3	REFERENCES.	1-3
1.4	DEFINITION OF TERMS	1-3
1.5	ACVC TEST CLASSES	1-4
CHAPTER 2	CONFIGURATION INFORMATION	
2.1	CONFIGURATION TESTED.	2-1
2.2	IMPLEMENTATION CHARACTERISTICS.	2-2
CHAPTER 3	TEST INFORMATION	
3.1	TEST RESULTS.	3-1
3.2	SUMMARY OF TEST RESULTS BY CLASS.	3-1
3.3	SUMMARY OF TEST RESULTS BY CHAPTER.	3-2
3.4	WITHDRAWN TESTS	3-2
3.5	INAPPLICABLE TESTS.	3-2
3.6	TEST, PROCESSING, AND EVALUATION MODIFICATIONS.	3-5
3.7	ADDITIONAL TESTING INFORMATION.	3-6
3.7.1	Prevalidation	3-6
3.7.2	Test Method	3-6
3.7.3	Test Site	3-8
APPENDIX A	DECLARATION OF CONFORMANCE	
APPENDIX B	APPENDIX F OF THE Ada STANDARD	
APPENDIX C	TEST PARAMETERS	
APPENDIX D	WITHDRAWN TESTS	
APPENDIX E	COMPILER OPTIONS AS SUPPLIED BY IBM	

CHAPTER 1

INTRODUCTION

This Validation Summary Report (VSR) describes the extent to which a specific Ada compiler conforms to the Ada Standard, ANSI/MIL-STD-1815A. This report explains all technical terms used within it and thoroughly reports the results of testing this compiler using the Ada Compiler Validation Capability (ACVC). An Ada compiler must be implemented according to the Ada Standard, and any implementation-dependent features must conform to the requirements of the Ada Standard. The Ada Standard must be implemented in its entirety, and nothing can be implemented that is not in the Standard.

Even though all validated Ada compilers conform to the Ada Standard, it must be understood that some differences do exist between implementations. The Ada Standard permits some implementation dependencies--for example, the maximum length of identifiers or the maximum values of integer types. Other differences between compilers result from the characteristics of particular operating systems, hardware, or implementation strategies. All the dependencies observed during the process of testing this compiler are given in this report.

The information in this report is derived from the test results produced during validation testing. The validation process includes submitting a suite of standardized tests, the ACVC, as inputs to an Ada compiler and evaluating the results. The purpose of validating is to ensure conformity of the compiler to the Ada Standard by testing that the compiler properly implements legal language constructs and that it identifies and rejects illegal language constructs. The testing also identifies behavior that is implementation-dependent but is permitted by the Ada Standard. Six classes of tests are used. These tests are designed to perform checks at compile time, at link time, and during execution.

INTRODUCTION

1.1 PURPOSE OF THIS VALIDATION SUMMARY REPORT

This VSR documents the results of the validation testing performed on an Ada compiler. Testing was carried out for the following purposes:

- . To attempt to identify any language constructs supported by the compiler that do not conform to the Ada Standard
- . To attempt to identify any language constructs not supported by the compiler but required by the Ada Standard
- . To determine that the implementation-dependent behavior is allowed by the Ada Standard

Testing of this compiler was conducted by SofTech, Inc. under the direction of the AVF according to procedures established by the Ada Joint Program Office and administered by the Ada Validation Organization (AVO). On-site testing was completed 29 November 1989 at Rockville MD.

1.2 USE OF THIS VALIDATION SUMMARY REPORT

Consistent with the national laws of the originating country, the AVO may make full and free public disclosure of this report. In the United States, this is provided in accordance with the "Freedom of Information Act" (5 U.S.C.#552). The results of this validation apply only to the computers, operating systems, and compiler versions identified in this report.

The organizations represented on the signature page of this report do not represent or warrant that all statements set forth in this report are accurate and complete, or that the subject compiler has no nonconformities to the Ada Standard other than those presented. Copies of this report are available to the public from:

Ada Information Clearinghouse
Ada Joint Program Office
OUSDRE
The Pentagon, Rm 3D-139 (Fern Street)
Washington DC 20301-3081

or from:

Ada Validation Facility
ASD/SCOL
Wright-Patterson AFB OH 45433-6503

INTRODUCTION

Questions regarding this report or the validation test results should be directed to the AVF listed above or to:

Ada Validation Organization
Institute for Defense Analyses
1801 North Beauregard Street
Alexandria VA 22311

1.3 REFERENCES

1. Reference Manual for the Ada Programming Language, ANSI/MIL-STD-1815A, February 1983 and ISO 8652-1987.
2. Ada Compiler Validation Procedures, Version 2.0, Ada Joint Program Office, May 1989.
3. Ada Compiler Validation Capability Implementers' Guide, SofTech, Inc., December 1986.
4. Ada Compiler Validation Capability User's Guide, December 1986.

1.4 DEFINITION OF TERMS

ACVC	The Ada Compiler Validation Capability. The set of Ada programs that tests the conformity of an Ada compiler to the Ada programming language.
Ada Commentary	An Ada Commentary contains all information relevant to the point addressed by a comment on the Ada Standard. These comments are given a unique identification number having the form AI-ddddd.
Ada Standard	ANSI/MIL-STD-1815A, February 1983 and ISO 8652-1987.
Applicant	The agency requesting validation.
AVF	The Ada Validation Facility. The AVF is responsible for conducting compiler validations according to procedures contained in the <u>Ada Compiler Validation Procedures</u> .
AVO	The Ada Validation Organization. The AVO has oversight authority over all AVF practices for the purpose of maintaining a uniform process for validation of Ada compilers. The AVO provides administrative and technical support for Ada validations to ensure consistent practices.
Compiler	A processor for the Ada language. In the context of this report, a compiler is any language processor, including cross-compilers, translators, and interpreters.

INTRODUCTION

Failed test	An ACVC test for which the compiler generates a result that demonstrates nonconformity to the Ada Standard.
Host	The computer on which the compiler resides.
Inapplicable test	An ACVC test that uses features of the language that a compiler is not required to support or may legitimately support in a way other than the one expected by the test.
Passed test	An ACVC test for which a compiler generates the expected result.
Target	The computer for which a compiler generates code.
Test	A program that checks a compiler's conformity regarding a particular feature or a combination of features to the Ada Standard. In the context of this report, the term is used to designate a single test, which may comprise one or more files.
Withdrawn test	An ACVC test found to be incorrect and not used to check conformity to the Ada Standard. A test may be incorrect because it has an invalid test objective, fails to meet its test objective, or contains illegal or erroneous use of the language.

1.5 ACVC TEST CLASSES

Conformity to the Ada Standard is measured using the ACVC. The ACVC contains both legal and illegal Ada programs structured into six test classes: A, B, C, D, E, and L. The first letter of a test name identifies the class to which it belongs. Class A, C, D, and E tests are executable, and special program units are used to report their results during execution. Class B tests are expected to produce compilation errors. Class L tests are expected to produce compilation or link errors because of the way in which a program library is used at link time.

Class A tests ensure the successful compilation of legal Ada programs with certain language constructs which cannot be verified at compile time. There are no explicit program components in a Class A test to check semantics. For example, a Class A test checks that reserved words of another language (other than those already reserved in the Ada language) are not treated as reserved words by an Ada compiler. A Class A test is passed if no errors are detected at compile time and the program executes to produce a PASSED message.

Class B tests check that a compiler detects illegal language usage. Class B tests are not executable. Each test in this class is compiled and the resulting compilation listing is examined to verify that every syntax or semantic error in the test is detected. A Class B test is passed if every illegal construct that it contains is detected by the compiler.

INTRODUCTION

Class C tests check the run time system to ensure that legal Ada programs can be correctly compiled and executed. Each Class C test is self-checking and produces a PASSED, FAILED, or NOT APPLICABLE message indicating the result when it is executed.

Class D tests check the compilation and execution capacities of a compiler. Since there are no capacity requirements placed on a compiler by the Ada Standard for some parameters--for example, the number of identifiers permitted in a compilation or the number of units in a library--a compiler may refuse to compile a Class D test and still be a conforming compiler. Therefore, if a Class D test fails to compile because the capacity of the compiler is exceeded, the test is classified as inapplicable. If a Class D test compiles successfully, it is self-checking and produces a PASSED or FAILED message during execution.

Class E tests are expected to execute successfully and check implementation-dependent options and resolutions of ambiguities in the Ada Standard. Each Class E test is self-checking and produces a NOT APPLICABLE, PASSED, or FAILED message when it is compiled and executed. However, the Ada Standard permits an implementation to reject programs containing some features addressed by Class E tests during compilation. Therefore, a Class E test is passed by a compiler if it is compiled successfully and executes to produce a PASSED message, or if it is rejected by the compiler for an allowable reason.

Class L tests check that incomplete or illegal Ada programs involving multiple, separately compiled units are detected and not allowed to execute. Class L tests are compiled separately and execution is attempted. A Class L test passes if it is rejected at link time--that is, an attempt to execute the main program must generate an error message before any declarations in the main program or any units referenced by the main program are elaborated. In some cases, an implementation may legitimately detect errors during compilation of the test.

Two library units, the package REPORT and the procedure CHECK_FILE, support the self-checking features of the executable tests. The package REPORT provides the mechanism by which executable tests report PASSED, FAILED, or NOT APPLICABLE results. It also provides a set of identity functions used to defeat some compiler optimizations allowed by the Ada Standard that would circumvent a test objective. The procedure CHECK_FILE is used to check the contents of text files written by some of the Class C tests for chapter 14 of the Ada Standard. The operation of REPORT and CHECK_FILE is checked by a set of executable tests. These tests produce messages that are examined to verify that the units are operating correctly. If these units are not operating correctly, then the validation is not attempted.

The text of each test in the ACVC follows conventions that are intended to ensure that the tests are reasonably portable without modification. For example, the tests make use of only the basic set of 55 characters, contain lines with a maximum length of 72 characters, use small numeric values, and place features that may not be supported by all implementations in separate tests. However, some tests contain values that require the test to be

INTRODUCTION

customized according to implementation-specific values--for example, an illegal file name. A list of the values used for this validation is provided in Appendix C.

A compiler must correctly process each of the tests in the suite and demonstrate conformity to the Ada Standard by either meeting the pass criteria given for the test or by showing that the test is inapplicable to the implementation. The applicability of a test to an implementation is considered each time the implementation is validated. A test that is inapplicable for one validation is not necessarily inapplicable for a subsequent validation. Any test that was determined to contain an illegal language construct or an erroneous language construct is withdrawn from the ACVC and, therefore, is not used in testing a compiler. The tests withdrawn at the time of this validation are given in Appendix D.

CHAPTER 2

CONFIGURATION INFORMATION

2.1 CONFIGURATION TESTED

The candidate compilation system for this validation was tested under the following configuration:

Compiler: The IBM Development System for the Ada Language
AIX/RT Follow-on, Version 1.1

ACVC Version: 1.10

Certificate Number: 891129W1.10198

Host Computer:

Machine: IBM RT Follow-on

Operating System: AIX
Version 3.1

Memory Size: 16 Megabytes

Target Computer:

Machine: IBM RT Follow-on

Operating System: AIX
Version 3.1

Memory Size: 16 Megabytes

CONFIGURATION INFORMATION

2.2 IMPLEMENTATION CHARACTERISTICS

One of the purposes of validating compilers is to determine the behavior of a compiler in those areas of the Ada Standard that permit implementations to differ. Class D and E tests specifically check for such implementation differences. However, tests in other classes also characterize an implementation. The tests demonstrate the following characteristics:

a. Capacities.

- (1) The compiler correctly processes a compilation containing 723 variables in the same declarative part. (See test D29002K.)
- (2) The compiler correctly processes tests containing loop statements nested to 65 levels. (See tests D55A03A..H (8 tests).)
- (3) The compiler correctly processes tests containing block statements nested to 65 levels. (See test D56001B.)
- (4) The compiler correctly processes tests containing recursive procedures separately compiled as subunits nested to 17 levels. (See tests D64005E..G (3 tests).)

b. Predefined types.

- (1) This implementation supports the additional predefined types `SHORT_INTEGER` and `LONG_FLOAT` in package `STANDARD`. (See tests B86001T..Z (7 tests).)

c. Expression evaluation.

The order in which expressions are evaluated and the time at which constraints are checked are not defined by the language. While the ACVC tests do not specifically attempt to determine the order of evaluation of expressions, test results indicate the following:

- (1) Some of the default initialization expressions for record components are evaluated before any value is checked for membership in a component's subtype. (See test C32117A.)
- (2) Assignments for subtypes are performed with the same precision as the base type. (See test C35712B.)
- (3) This implementation uses no extra bits for extra precision and uses no extra bits for extra range. (See test C35903A.)

CONFIGURATION INFORMATION

- (4) Sometimes `NUMERIC_ERROR` is raised when an integer literal operand in a comparison or membership test is outside the range of the base type. (See test C45232A.)
- (5) `NUMERIC_ERROR` is raised when a literal operand in a fixed-point comparison or membership test is outside the range of the base type. (See test C45252A.)
- (6) Underflow is gradual. (See tests C45524A..Z (26 tests).)

d. Rounding.

The method by which values are rounded in type conversions is not defined by the language. While the ACVC tests do not specifically attempt to determine the method of rounding, the test results indicate the following:

- (1) The method used for rounding to integer is round away from zero. (See tests C46012A..Z (26 tests).)
- (2) The method used for rounding to longest integer is round away from zero. (See tests C46012A..Z (26 tests).)
- (3) The method used for rounding to integer in static universal real expressions is round away from zero. (See test C4A014A.)

e. Array types.

An implementation is allowed to raise `NUMERIC_ERROR` or `CONSTRAINT_ERROR` for an array having a `'LENGTH` that exceeds `STANDARD.INTEGER'LAST` and/or `SYSTEM.MAX_INT`.

For this implementation:

- (1) Declaration of an array type or subtype declaration with more than `SYSTEM.MAX_INT` components raises no exception. (See test C36003A.)
- (2) `NUMERIC_ERROR` is raised when an array type with `INTEGER'LAST + 2` components is with each component being a null array declared. (See test C36202A.)
- (3) `NUMERIC_ERROR` is raised when an array type with `SYSTEM.MAX_INT + 2` components with each component being a null array is declared. (See test C36202B.)
- (4) A packed `BOOLEAN` array having a `'LENGTH` exceeding `INTEGER'LAST` raises `NUMERIC_ERROR` when the array type is declared. (See test C52103X.)

CONFIGURATION INFORMATION

- (5) A packed two-dimensional BOOLEAN array with more than INTEGER'LAST components raises NUMERIC_ERROR when the array type is declared. (See test C52104Y.)
- (6) A null array with one dimension of length greater than INTEGER'LAST may raise NUMERIC_ERROR or CONSTRAINT_ERROR either when declared or assigned. Alternatively, an implementation may accept the declaration. However, lengths must match in array slice assignments. This implementation raises NUMERIC_ERROR when the array type is declared. (See test E52103Y.)
- (7) In assigning one-dimensional array types, the expression is evaluated in its entirety before CONSTRAINT_ERROR is raised when checking whether the expression's subtype is compatible with the target's subtype. (See test C52013A.)
- (8) In assigning two-dimensional array types, the expression is not evaluated in its entirety before CONSTRAINT_ERROR is raised when checking whether the expression's subtype is compatible with the target's subtype. (See test C52013A.)

f. Discriminated types.

- (1) In assigning record types with discriminants, the expression is evaluated in its entirety before CONSTRAINT_ERROR is raised when checking whether the expression's subtype is compatible with the target's subtype. (See test C52013A.)

g. Aggregates.

- (1) In the evaluation of a multi-dimensional aggregate, index subtype checks are made as choices are evaluated. (See tests C43207A and C43207B.)
- (2) In the evaluation of an aggregate containing subaggregates, not all choices are evaluated before being checked for identical bounds. (See test E43212B.)
- (3) CONSTRAINT_ERROR is raised after all choices are evaluated when a bound in a non-null range of a non-null aggregate does not belong to an index subtype. (See test E43211B.)

h. Pragmas.

- (1) The pragma INLINE is not supported for procedures or functions. (See tests LA3004A..B (2 tests), EA3004C..D (2 tests), and CA3004E..F (2 tests).)

CONFIGURATION INFORMATION

i. Generics.

- (1) Generic library subprogram specifications and bodies can be compiled in separate compilations. (See test CA1012A.)
- (2) If a generic unit body or one of its subunits is compiled or recompiled after the generic unit is instantiated, the unit instantiating the generic is made obsolete. The obsolescence is recognized at binding time, and the binding is stopped. (See tests CA2009C, CA2009F, BC3204C, and BC3205D.)
- (3) Generic unit bodies and their subunits can be compiled in separate compilations. (See test CA3011A.)

j. Input and output.

- (1) The package `DIRECT_IO` can be instantiated with unconstrained array types and record types with discriminants without defaults. (See tests AE2101H, EE2401D, and EE2401G.)
- (2) The package `SEQUENTIAL_IO` can be instantiated with unconstrained array types and record types with discriminants without defaults. (See tests AE2101C, EE2201D, and EE2201E.)
- (3) Modes `IN FILE` and `OUT FILE` are supported for `SEQUENTIAL_IO`. (See tests CE2102D..E (2 tests), CE2102N, and CE2102P.)
- (4) Modes `IN FILE`, `OUT FILE`, and `INOUT FILE` are supported for `DIRECT_IO`. (See tests CE2102F, CE2102I..J (2 tests), CE2102R, CE2102T, and CE2102V.)
- (5) Modes `IN FILE` and `OUT FILE` are supported for text files. (See tests CE3102E and CE3102I..K (3 tests).)
- (6) `RESET` and `DELETE` operations are supported for `SEQUENTIAL_IO`. (See tests CE2102G and CE2102X.)
- (7) `RESET` and `DELETE` operations are supported for `DIRECT_IO`. (See tests CE2102K and CE2102Y.)
- (8) `RESET` and `DELETE` operations are supported for text files. (See tests CE3102F..G (2 tests), CE3104C, CE3110A, and CE3114A.)
- (9) Overwriting to a sequential file does not truncate the file. (See test CE2208B.)
- (10) Temporary sequential files are given names and deleted when closed. (See test CE2108A.)

CONFIGURATION INFORMATION

- (11) Temporary direct files are given names and deleted when closed. (See test CE2108C.)
- (12) Temporary text files are given names and deleted when closed. (See test CE3112A.)
- (13) More than one internal file can be associated with each external file for sequential files when writing or reading. (See tests CE2107A..E (5 tests), CE2102L, CE2110B, and CE2111D.)
- (14) More than one internal file can be associated with each external file for direct files when writing or reading. (See tests CE2107F..H (3 tests), CE2110D, and CE2111H.)
- (15) More than one internal file can be associated with each external file for text files when writing or reading. (See tests CE3111A..E (5 tests), CE3114B, and CE3115A.)

CHAPTER 3
TEST INFORMATION

3.1 TEST RESULTS

Version 1.10 of the ACVC comprises 3717 tests. When this compiler was tested, 44 tests had been withdrawn because of test errors. The AVF determined that 293 tests were inapplicable to this implementation. All inapplicable tests were processed during validation testing except for 201 executable tests that use floating-point precision exceeding that supported by the implementation. Modifications to the code, processing, or grading for 12 tests were required to successfully demonstrate the test objective.

The AVF concludes that the testing results demonstrate acceptable conformity to the Ada Standard.

3.2 SUMMARY OF TEST RESULTS BY CLASS

RESULT	TEST CLASS						TOTAL
	A	B	C	D	E	L	
Passed	129	1129	2035	17	26	44	3380
Inapplicable	0	9	280	0	2	2	293
Withdrawn	1	2	35	0	6	0	44
TOTAL	130	1140	2350	17	34	46	3717

TEST INFORMATION

3.3 SUMMARY OF TEST RESULTS BY CHAPTER

RESULT	CHAPTER													TOTAL
	2	3	4	5	6	7	8	9	10	11	12	13	14	
Passed	198	573	544	245	172	99	161	332	129	36	250	341	300	3380
Inappl	14	76	136	3	0	0	5	0	8	0	2	28	21	293
Wdrn	1	1	0	0	0	0	0	2	0	0	1	35	4	44
TOTAL	213	650	680	248	172	99	166	334	137	36	253	404	325	3717

3.4 WITHDRAWN TESTS

The following 44 tests were withdrawn from ACVC Version 1.10 at the time of this validation:

E28005C	A39005G	B97102E	C97116A	BC3009B	CD2A62D
CD2A63A	CD2A63B	CD2A63C	CD2A63D	CD2A66A	CD2A66B
CD2A66C	CD2A66D	CD2A73A	CD2A73B	CD2A73C	CD2A73D
CD2A76A	CD2A76B	CD2A76C	CD2A76D	CD2A81G	CD2A83G
CD2A84M	CD2A84N	CD2B15C	CD2D11B	CD5007B	CD50110
ED7004B	ED7005C	ED7005D	ED7006C	ED7006D	CD7105A
CD7203B	CD7204B	CD7205C	CD7205D	CE2107I	CE3111C
CE3301A	CE3411B				

See Appendix D for the reason that each of these tests was withdrawn.

3.5 INAPPLICABLE TESTS

Some tests do not apply to all compilers because they make use of features that a compiler is not required by the Ada Standard to support. Others may depend on the result of another test that is either inapplicable or withdrawn. The applicability of a test to an implementation is considered each time a validation is attempted. A test that is inapplicable for one validation attempt is not necessarily inapplicable for a subsequent attempt. For this validation attempt, 293 tests were inapplicable for the reasons indicated:

- a. The following 201 tests are not applicable because they have floating-point type declarations requiring more digits than `SYSTEM.MAX_DIGITS`:

C24113L..Y (14 tests)	C35705L..Y (14 tests)
C35706L..Y (14 tests)	C35707L..Y (14 tests)
C35708L..Y (14 tests)	C35802L..Z (15 tests)

TEST INFORMATION

C45241L..Y (14 tests)	C45321L..Y (14 tests)
C45421L..Y (14 tests)	C45521L..Z (15 tests)
C45524L..Z (15 tests)	C45621L..Z (15 tests)
C45641L..Y (14 tests)	C46012L..Z (15 tests)

- b. C35508I, C35508J, C35508M, and C35508N are not applicable because they include enumeration representation clauses for BOOLEAN types in which the representation values are other than (FALSE => 0, TRUE => 1). Under the terms of AI-00325, this implementation is not required to support such representation clauses.
- c. C35702A and B86001T are not applicable because this implementation supports no predefined type SHORT_FLOAT.
- d. The following 16 tests are not applicable because this implementation does not support a predefined type LONG_INTEGER:

C45231C	C45304C	C45502C	C45503C	C45504C
C45504F	C45611C	C45613C	C45614C	C45631C
C45632C	B52004D	C55B07A	B55B09C	B86001W
CD7101F				
- e. C45231D, B86001X, and CD7101G are not applicable because this implementation does not support any predefined integer type with a name other than INTEGER, LONG_INTEGER, or SHORT_INTEGER.
- f. C45531M..P (4 tests) and C45532M..P (4 tests) are not applicable because the value of SYSTEM.MAX_MANTISSA is less than 48.
- g. B86001Y is not applicable because this implementation supports no predefined fixed-point type other than DURATION.
- h. B86001Z is not applicable because this implementation supports no predefined floating-point type with a name other than FLOAT, LONG_FLOAT, or SHORT_FLOAT.
- i. CA2009C, CA2009F, BC3204C, and BC3205D are not applicable because this implementation does not support separate compilation of generic specifications, bodies, and subunits, if an instantiation is given before compilation of its bodies or subunits. The created dependency is detected at bind time.
- j. LA3004A, LA3004B, EA3004C, EA3004D, CA3004E, and CA3004F are not applicable because this implementation does not support pragma INLINE.
- k. CD1009C, CD2A41A..B (2 tests), CD2A41E, and CD2A42A..J (10 tests) are not applicable because this implementation requires a minimum of 32 bits to represent floating point types.
- l. CD2A61I and CD2A61J are not applicable because this implementation does not support size clauses for array types, which imply compression, with component types of composite or floating point

TEST INFORMATION

types. This implementation requires an explicit size clause on the component type.

- m. CD2A84B..I (8 tests) and CD2A84K..L (2 tests) are not applicable because this implementation does not support access types of less than 32 bits.
- n. CE2102D is inapplicable because this implementation supports CREATE with IN_FILE mode for SEQUENTIAL_IO.
- o. CE2102E is inapplicable because this implementation supports CREATE with OUT_FILE mode for SEQUENTIAL_IO.
- p. CE2102F is inapplicable because this implementation supports CREATE with INOUT_FILE mode for DIRECT_IO.
- q. CE2102I is inapplicable because this implementation supports CREATE with IN_FILE mode for DIRECT_IO.
- r. CE2102J is inapplicable because this implementation supports CREATE with OUT_FILE mode for DIRECT_IO.
- s. CE2102N is inapplicable because this implementation supports OPEN with IN_FILE mode for SEQUENTIAL_IO.
- t. CE2102O is inapplicable because this implementation supports RESET with IN_FILE mode for SEQUENTIAL_IO.
- u. CE2102P is inapplicable because this implementation supports OPEN with OUT_FILE mode for SEQUENTIAL_IO.
- v. CE2102Q is inapplicable because this implementation supports RESET with OUT_FILE mode for SEQUENTIAL_IO.
- w. CE2102R is inapplicable because this implementation supports OPEN with INOUT_FILE mode for DIRECT_IO.
- x. CE2102S is inapplicable because this implementation supports RESET with INOUT_FILE mode for DIRECT_IO.
- y. CE2102T is inapplicable because this implementation supports OPEN with IN_FILE mode for DIRECT_IO.
- z. CE2102U is inapplicable because this implementation supports RESET with IN_FILE mode for DIRECT_IO.
- aa. CE2102V is inapplicable because this implementation supports OPEN with OUT_FILE mode for DIRECT_IO.
- ab. CE2102W is inapplicable because this implementation supports RESET with OUT_FILE mode for DIRECT_IO.

TEST INFORMATION

- ac. CE3102E is inapplicable because this implementation supports CREATE with IN_FILE mode for text files.
- ad. CE3102F is inapplicable because this implementation supports RESET for text files.
- ae. CE3102G is inapplicable because this implementation supports deletion of an external file for text files.
- af. CE3102I is inapplicable because this implementation supports CREATE with OUT_FILE mode for text files.
- ag. CE3102J is inapplicable because this implementation supports OPEN with IN_FILE mode for text files.
- ah. CE3102K is inapplicable because this implementation supports OPEN with OUT_FILE mode for text files.

3.6 TEST, PROCESSING, AND EVALUATION MODIFICATIONS

It is expected that some tests will require modifications of code, processing, or evaluation in order to compensate for legitimate implementation behavior. Modifications are made by the AVF in cases where legitimate implementation behavior prevents the successful completion of an (otherwise) applicable test. Examples of such modifications include: adding a length clause to alter the default size of a collection; splitting a Class B test into subtests so that all errors are detected; and confirming that messages produced by an executable test demonstrate conforming behavior that wasn't anticipated by the test (such as raising one exception instead of another).

Modifications were required for 12 tests.

The following tests were split because syntax errors at one point resulted in the compiler not detecting other errors in the test:

BA3006A BA3006B BA3007B BA3008A BA3013A

C34005G, C34005J, and C34006D required evaluation modifications because the tests include some comparisons that use the 'SIZE attribute under assumptions that are not fully supported by the Ada Standard and are subject to ARG review. Thus, the AVO ruled that an implementation is considered to have passed these tests if the only REPORT.FAILED output is because of various 'SIZE checks. This implementation produced the messages "INCORRECT TYPE'SIZE", "INCORRECT OBJECT'SIZE, and "INCORRECT BASE'SIZE" for C34005G and the message "INCORRECT TYPE'SIZE" for C34006D.

C52008B required modification because this implementation does not support a record type with four discriminants of type integer having default values. The size of this object exceeds the maximum object size of this

TEST INFORMATION

implementation and NUMERIC ERROR is raised. A subtype S_INTEGER was declared of type integer range 0..127 and the discriminants were declared to be of type S_INTEGER.

CD2C11A and CD2C11B required modification because this implementation requires a greater STORAGE_SIZE than specified by the test. The amount of storage specified by the STORAGE_SIZE representation was changed from 1024 to 2048.

CE3804G required evaluation modifications because the value "-3.525" in the test is not a model number of the declared type and thus the equality test may legitimately fail, yielding the message "WIDTH CHARACTERS NOT READ - FLOAT 3". In this implementation, the algorithm used by FLOAT_IO to convert a character string to a hardware floating point value and the algorithm used by the code generator to convert a UNIVERSAL_REAL from the low form to a hardware floating point value do not yield the same result.

3.7 ADDITIONAL TESTING INFORMATION

3.7.1 Prevalidation

Prior to validation, a set of test results for ACVC Version 1.10 produced by The IBM Development System for the Ada Language AIX/RT Follow-on, Version 1.1, compiler was submitted to the AVF by the applicant for review. Analysis of these results demonstrated that the compiler successfully passed all applicable tests, and the compiler exhibited the expected behavior on all inapplicable tests.

3.7.2 Test Method

Testing of The IBM Development System for the Ada Language AIX/RT Follow-on, Version 1.1, compiler using ACVC Version 1.10 was conducted on-site by a validation team from the AVF. The configuration in which the testing was performed is described by the following designations of hardware and software components:

Host computer:	IBM RT Follow-on
Host operating system:	AIX, Version 3.1
Target computer:	IBM RT Follow-on
Target operating system:	AIX, Version 3.1
Compiler:	The IBM Development System for the Ada Language AIX/RT Follow-on, Version 1.1

A set of diskettes containing all tests except for withdrawn tests and tests requiring unsupported floating-point precisions was taken on-site by the validation team for processing. Tests that make use of implementation-specific values were customized before being written to the diskettes. Tests requiring modifications during the prevalidation testing were included in their modified form on the diskettes.

TEST INFORMATION

The contents of the diskettes were not loaded directly onto the host computer. Since the host processor does not support a DOS capability, the DOS diskettes were loaded onto a PC-AT. The DOS files were transferred, using kermit, into an AIX directory on a 6150 Model 125 RT. The AIX subdirectory structure mapped directly into the DOS subdirectory structure contained within the DOS diskettes supplied by AVF.

After all the files were transferred onto the RT, a script was run which changed the 8-character DOS names to 9-character AIX names which the ACVC test scripts recognize. Then the ACVC tests were transferred to diskettes using the AIX "backup" command and from these diskettes to the IBM RT Follow-on machines using the AIX "restore" command.

After the test files were loaded to disk, the full set of tests was compiled, linked, and all executable tests were run on the IBM RT Follow-on. Results were transferred to the RT by means of diskettes, using the AIX "backup" and "restore" commands. Results from the B2 tests were printed directly from the RT using an IBM Proprinter III XL. The rest of the results were transferred to the IBM 370 system using the IBM Token Ring and the TCP/IP Communications Protocol. They were printed from the IBM 370 using a 3800 printer.

The compiler was tested using command scripts provided by International Business Machines Corporation and reviewed by the validation team. The compiler was tested using all the following option settings. See Appendix E for a complete listing of the compiler options for this implementation. The following list of compiler options includes those options which were invoked by default:

- cg_debug Specify that code generator debugging information should not be produced.
- cg_optimize Specify that the code generator may not perform optimizations that may interfere with debug tools.
- optimize Turn off the global optimizer.
- monitor Operate silently, without progress messages.
- +object Produce object code output (as opposed to assembly code).
- +virtual=3000 Specifies 3000 as the number of virtual pages used by the virtual space manager.
- +listing Generate source listings.
- +bind Produce an executable from previously compiled code. The Ada name of the main unit must also be specified on the command line.

TEST INFORMATION

Tests were compiled, linked, and executed (as appropriate) using 3 computers. Test output, compilation listings, and job logs were captured on diskettes and archived at the AVF. The listings examined on-site by the validation team were also archived.

3.7.3 Test Site

Testing was conducted at Rockville MD and was completed on 29 November 1989.

APPENDIX A

DECLARATION OF CONFORMANCE

International Business Machines Corporation has submitted the following Declaration of Conformance concerning The IBM Development System for the Ada Language AIX/RT Follow-on, Version 1.1 compiler.

DECLARATION OF CONFORMANCE

Compiler Implementer: International Business Machines Corporation
Ada Validation Facility: ASD/SCEL, Wright-Patterson AFB OH 45433-6503
Ada Compiler Validation Capability (ACVC) Version : 1.10

Base Configuration

Base Compiler Name: IBM Development System for the Ada Language AIX/RT Follow-on
Version: 1.1

Host Architecture ISA: IBM RT Follow-on OS&VER#: AIX,3.1
Target Architecture ISA: IBM RT Follow-on OS&VER#: AIX,3.1

Implementer's Declaration

I, the undersigned, representing the IBM Corporation have implemented no deliberate extensions to the Ada Language Standard ANSI/MIL-STD-1815A in the compiler listed in this declaration. I declare that the IBM Corporation is the owner of record of the object code of the Ada language compiler listed above and, as such, is responsible for maintaining said compiler in conformance to ANSI/MIL-STD-1815A. All certificates and registrations for the Ada language compiler listed in this declaration shall be made only in the owner's corporate name.

B.K. North
International Business Machines Corporation
B.K. North, Manager, Ada Technology and Performance

Date: 10/23/89

Owner's Declaration

I, the undersigned, representing the IBM Corporation take full responsibility for implementation and maintenance of the Ada compiler listed above, and agree to the public disclosure of the final Validation Summary Report. I declare that the Ada language compiler listed, and the host/target performance are in compliance with the Ada Language Standard ANSI/MIL-STD-1851A.

Manager, CRP
< Owner's Signature and Title >
< To be completed upon availability of the VSR >

Date: 5.12.19

APPENDIX B

APPENDIX F OF THE Ada STANDARD

The only allowed implementation dependencies correspond to implementation-dependent pragmas, to certain machine-dependent conventions as mentioned in Chapter 13 of the Ada Standard, and to certain allowed restrictions on representation clauses. The implementation-dependent characteristics of The IBM Development System for the Ada Language AIX/RT Follow-on, Version 1.1, compiler, as described in this Appendix, are provided by International Business Machines Corporation. Unless specifically noted otherwise, references in this Appendix are to compiler documentation and not to this report. Implementation-specific portions of the package STANDARD, which are not a part of Appendix F, are:

package STANDARD is

...

type INTEGER is range -2_147_483_648 .. 2_147_483_647;

type SHORT_INTEGER is range -32_768 .. 32_767;

type FLOAT is digits 6 range -3.40282E+38 .. 3.40282E+38;

type LONG_FLOAT is digits 15 range -1.79769227807399E+308 ..
1.79769227807399E+308;

type DURATION is delta 2**(-14) range -86_400.0 .. 86_400.0;

...

end STANDARD;

Implementation-Defined Pragmas

Implementation dependent pragmas are:

- Pragma IMAGES (enumeration_type, <immediate> | <deferred>);

Generates a table of images for the enumeration type. 'deferred' causes the table to be generated only if the enumeration type is in a compilation unit.

- Pragma LINKNAME (<pragma INTERFACE subprogram name> , <linkname>);

When used in conjunction with pragma INTERFACE, provides access to any routine whose name can be specified by an Ada string literal.

Predefined Pragmas

Supported pragmas are INTERFACE, ELABORATE, SUPPRESS, PACK, PAGE, LIST, and PRIORITY.

All pragmas have conventional meanings except LIST, which suppresses listings prior to Pragma LIST(ON) regardless of the user request.

Pragma INTERFACE supports C and Assembly.

Unrecognized and unsupported pragmas are ignored with an appropriate warning message.

Representation Clauses

Supported representation clauses include:

1. Length clause
2. Enumeration Representation Clauses, except for boolean types
3. Record Representation Clause
4. Address Clause
5. Interrupt Support

Record representation clauses are aligned on 32-bit boundaries.

Restrictions on Representation Clauses

The hardware requires a minimum of 32 bits for floating point and address types.

Restrictions On Unchecked Conversion

Unchecked_Conversion between two types (or subtypes) A and B is permitted provided that A and B are the same static size, and neither A nor B are private.

Package System

PACKAGE System IS

TYPE Address is access integer;

TYPE Name IS (TeleSoft_Ada);

System_Name : CONSTANT name := TeleSoft_Ada;

Storage_Unit : CONSTANT := 8;

Memory_Size : CONSTANT := 1024*1024*256;

-- System-Dependent Named Numbers:

Min_Int : CONSTANT := -(2 ** 31);

Max_Int : CONSTANT := (2 ** 31) - 1;

Max_Digits : CONSTANT := 15;

Max_Mantissa : CONSTANT := 31;

Fine_Delta : CONSTANT := 1.0 / (2 ** (Max_Mantissa));

Tick : CONSTANT := 0.00006;

-- Other System-Dependent Declarations

SUBTYPE Priority IS Integer RANGE 0 .. 255;

Max_Object_Size : CONSTANT := (32*1024)-1;

Max_Record_Count : CONSTANT := (32*1024)-1;

Max_Text_Io_Count : CONSTANT := 16*1024;

Max_Text_Io_Field : CONSTANT := 1000;

-- Other Types

TYPE Subprogram_Value is

Record

Entry_Point : Address;

Static_Base : Address;

End Record;

END System;

Representation Attributes

All define representation attributes shall be supported.

Convention Used For Generating Names

There are no implementation-generated names denoting implementation- dependent components. Names generated by the compiler shall not interfere with programmer-defined names.

Implementation Defined Characteristics For Input-Output Packages.

Sequential_IO, Direct_IO, and Text_IO are supported.

Low_Level_IO is not supported.

File names follow the conventions and restrictions of the target operating system, except that non-printing characters, like blank(' ') and asterisk('*') are disallowed.

In Text_IO, the type Field is defined as follows:

subtype Field is integer range 0..1000;

In Text_IO the type Count is defined as follows:

type Count is range 0..2_147_483_646;

Predefined Numeric Types

SHORT_INTEGER

'First	-32768
'Last	32767
'Size	16

INTEGER

'First	-2147483648
'Last	2147483647
'Size	32

FLOAT

'Machine_Overflows	True
'Machine_Rounds	True
'Machine_Radix	2
'Machine_Mantissa	24
'Machine_Emax	128
'Machine_Emin	-125
'Mantissa	21
'Digits	6
'Size	32
'Emax	84
'Safe_Emax	124
'Epsilon	9.53674E-07
'Safe_Large	2.12676E + 37
'Safe_Small	2.35099E-38
'Large	1.93428E + 25
'Small	2.58494E-26

LONG FLOAT

'Machine_Overflows	True
'Machine_Rounds	True
'Machine_Radix	2
'Machine_Mantissa	53
'Machine_Emax	1024
'Machine_Emin	-1024
'Mantissa	51
'Digits	15
'Size	64
'Emax	204
'Safe_Emax	1020
'Epsilon	1.05285E-15
'Safe_Large	1.0519E + 307
'Safe_Small	1.42138E-307
'Large	1.50700E + 61
'Small	2.03633E-61

DURATION

'Machine_Overflows	false
'Machine_Rounds	false
'Delta	2**(-14)
'First	-86400.0
'Last	86400.0

Restrictions on Machine Code Insertions

Machine code insertions are not supported.

Restrictions on Generics

Generic specifications, bodies and subunits may be divided into separate compilation units. The following restrictions apply for the instantiation of generics:

- The generic bodies and subunits must be compiled prior to the generic instantiation.
- The sublibrary containing the generic bodies and subunits must be visible at the time of the generic instantiation.

If either of the following conditions are not met, the compiler will generate a warning at compile time and fail during the binding of the main program.

APPENDIX C

TEST PARAMETERS

Certain tests in the ACVC make use of implementation-dependent values, such as the maximum length of an input line and invalid file names. A test that makes use of such values is identified by the extension .TST in its file name. Actual values to be substituted are represented by names that begin with a dollar sign. A value must be substituted for each of these names before the test is run. The values used for this validation are given below.

Name and Meaning	Value
\$ACC_SIZE An integer literal whose value is the number of bits sufficient to hold any value of an access type.	32
\$BIG_ID1 An identifier the size of the maximum input line length which is identical to \$BIG_ID2 except for the last character.	(1..199 => 'A', 200 => '1')
\$BIG_ID2 An identifier the size of the maximum input line length which is identical to \$BIG_ID1 except for the last character.	(1..199 => 'A', 200 => '2')
\$BIG_ID3 An identifier the size of the maximum input line length which is identical to \$BIG_ID4 except for a character near the middle.	(1..100 => 'A', 101 => '3', 102..200 => 'A')

TEST PARAMETERS

Name and Meaning	Value
\$BIG_ID4 An identifier the size of the maximum input line length which is identical to \$BIG_ID3 except for a character near the middle.	(1..100 => 'A', 101 => '4', 102..200 => 'A')
\$BIG_INT_LIT An integer literal of value 298 with enough leading zeroes so that it is the size of the maximum line length.	(1..197 => '0', 198..200 => "298")
\$BIG_REAL_LIT A universal real literal of value 690.0 with enough leading zeroes to be the size of the maximum line length.	(1..194 => '0', 195..200 => "69.0E1")
\$BIG_STRING1 A string literal which when catenated with \$BIG_STRING2 yields the image of \$BIG_ID1.	(1 => '"', 2..101 => 'A', 102 => '"')
\$BIG_STRING2 A string literal which when catenated to the end of \$BIG_STRING1 yields the image of \$BIG_ID1.	(1 => '"', 2..100 => 'A', 101 => '1', 102 => '"')
\$BLANKS A sequence of blanks twenty characters less than the size of the maximum line length.	(1..180 => ' ')
\$COUNT_LAST A universal integer literal whose value is TEXT_IO.COUNT'LAST.	2_147_483_646
\$DEFAULT_MEM_SIZE An integer literal whose value is SYSTEM.MEMORY_SIZE.	268_435_456
\$DEFAULT_STOR_UNIT An integer literal whose value is SYSTEM.STORAGE_UNIT.	8

TEST PARAMETERS

Name and Meaning	Value
\$DEFAULT_SYS_NAME The value of the constant SYSTEM.SYSTEM_NAME.	TELESOFT_ADA
\$DELTA_DOC A real literal whose value is SYSTEM.FINE_DELTA.	2#1.0#E-31
\$FIELD_LAST A universal integer literal whose value is TEXT_IO.FIELD'LAST.	1_000
\$FIXED_NAME The name of a predefined fixed-point type other than DURATION.	NO_SUCH_FIXED_TYPE
\$FLOAT_NAME The name of a predefined floating-point type other than FLOAT, SHORT_FLOAT, or LONG_FLOAT.	NO_SUCH_FLOAT_TYPE
\$GREATER_THAN_DURATION A universal real literal that lies between DURATION'BASE'LAST and DURATION'LAST or any value in the range of DURATION.	100_000.0
\$GREATER_THAN_DURATION_BASE_LAST A universal real literal that is greater than DURATION'BASE'LAST.	131_073.0
\$HIGH_PRIORITY An integer literal whose value is the upper bound of the range for the subtype SYSTEM.PRIORITY.	255
\$ILLEGAL_EXTERNAL_FILE_NAME1 An external file name which contains invalid characters.	BADCHAR*~/%
\$ILLEGAL_EXTERNAL_FILE_NAME2 An external file name which is too long.	/NONAME/DIRECTORY
\$INTEGER_FIRST A universal integer literal whose value is INTEGER'FIRST.	-2_147_483_648

TEST PARAMETERS

Name and Meaning	Value
\$INTEGER_LAST A universal integer literal whose value is INTEGER'LAST.	2_147_483_647
\$INTEGER_LAST_PLUS_1 A universal integer literal whose value is INTEGER'LAST + 1.	2_147_483_648
\$LESS_THAN_DURATION A universal real literal that lies between DURATION'BASE'FIRST and DURATION'FIRST or any value in the range of DURATION.	-100_000.0
\$LESS_THAN_DURATION_BASE_FIRST A universal real literal that is less than DURATION'BASE'FIRST.	-131_073.0
\$LOW_PRIORITY An integer literal whose value is the lower bound of the range for the subtype SYSTEM.PRIORITY.	0
\$MANTISSA_DOC An integer literal whose value is SYSTEM.MAX_MANTISSA.	31
\$MAX_DIGITS Maximum digits supported for floating-point types.	15
\$MAX_IN_LEN Maximum input line length permitted by the implementation.	200
\$MAX_INT A universal integer literal whose value is SYSTEM.MAX_INT.	2_147_483_647
\$MAX_INT_PLUS_1 A universal integer literal whose value is SYSTEM.MAX_INT+1.	2_147_483_648
\$MAX_LEN_INT_BASED_LITERAL A universal integer based literal whose value is 2#11# with enough leading zeroes in the mantissa to be \$MAX_IN_LEN long.	(1..2 => "2:", 3..197 => '0', 198..200 => "11:")

TEST PARAMETERS

Name and Meaning	Value
<p>\$MAX_LEN_REAL_BASED_LITERAL A universal real based literal whose value is 16:F.E: with enough leading zeroes in the mantissa to be \$MAX_IN_LEN long.</p>	(1..3 => "16:", 4..196 => '0', 197..200 => "F.E:")
<p>\$MAX_STRING_LITERAL A string literal of size \$MAX_IN_LEN, including the quote characters.</p>	(1 => '"', 2..199 => 'A', 200 => '"')
<p>\$MIN_INT A universal integer literal whose value is SYSTEM.MIN_INT.</p>	-2_147_483_648
<p>\$MIN_TASK_SIZE An integer literal whose value is the number of bits required to hold a task object which has no entries, no declarations, and "NULL;" as the only statement in its body.</p>	32
<p>\$NAME A name of a predefined numeric type other than FLOAT, INTEGER, SHORT_FLOAT, SHORT_INTEGER, LONG_FLOAT, or LONG_INTEGER.</p>	NO_SUCH_TYPE_AVAILABLE
<p>\$NAME_LIST A list of enumeration literals in the type SYSTEM.NAME, separated by commas.</p>	TELESOFT_ADA
<p>\$NEG_BASED_INT A based integer literal whose highest order nonzero bit falls in the sign bit position of the representation for SYSTEM.MAX_INT.</p>	16#FFFFFFFFE#
<p>\$NEW_MEM_SIZE An integer literal whose value is a permitted argument for pragma MEMORY_SIZE, other than \$DEFAULT_MEM_SIZE. If there is no other value, then use \$DEFAULT_MEM_SIZE.</p>	268_435_456

TEST PARAMETERS

Name and Meaning	Value
\$NEW_STOR_UNIT An integer literal whose value is a permitted argument for pragma STORAGE UNIT, other than \$DEFAULT_STOR_UNIT. If there is no other permitted value, then use value of SYSTEM.STORAGE_UNIT.	8
\$NEW_SYS_NAME A value of the type SYSTEM.NAME, other than \$DEFAULT_SYS_NAME. If there is only one value of that type, then use that value.	TELESOFT_ADA
\$TASK_SIZE An integer literal whose value is the number of bits required to hold a task object which has a single entry with one 'IN OUT' parameter.	32
\$TICK A real literal whose value is SYSTEM.TICK.	0.00006

APPENDIX D
WITHDRAWN TESTS

Some tests are withdrawn from the ACVC because they do not conform to the Ada Standard. The following 44 tests had been withdrawn at the time of validation testing for the reasons indicated. A reference of the form AI-ddddd is to an Ada Commentary.

- a. E28005C: This test expects that the string "-- TOP OF PAGE. --63" of line 204 will appear at the top of the listing page due to a pragma PAGE in line 203; but line 203 contains text that follows the pragma, and it is this text that must appear at the top of the page.
- b. A39005G: This test unreasonably expects a component clause to pack an array component into a minimum size (line 30).
- c. B97102E: This test contains an unintended illegality: a select statement contains a null statement at the place of a selective wait alternative (line 31).
- d. C97116A: This test contains race conditions, and it assumes that guards are evaluated indivisibly. A conforming implementation may use interleaved execution in such a way that the evaluation of the guards at lines 50 & 54 and the execution of task CHANGING OF THE GUARD results in a call to REPORT.FAILED at one of lines 52 or 56.
- e. BC3009B: This test wrongly expects that circular instantiations will be detected in several compilation units even though none of the units is illegal with respect to the units it depends on; by AI-00256, the illegality need not be detected until execution is attempted (line 95).
- f. CD2A62D: This test wrongly requires that an array object's size be no greater than 10 although its subtype's size was specified to be 40 (line 137).

WITHDRAWN TESTS

- g. CD2A63A..D, CD2A66A..D, CD2A73A..D, and CD2A76A..D (16 tests): These tests wrongly attempt to check the size of objects of a derived type (for which a 'SIZE length clause is given) by passing them to a derived subprogram (which implicitly converts them to the parent type (Ada standard 3.4:14)). Additionally, they use the 'SIZE length clause and attribute, whose interpretation is considered problematic by the WG9 ARG.
- h. CD2A81G, CD2A83G, CD2A84M..N, and CD50110 (5 tests): These tests assume that dependent tasks will terminate while the main program executes a loop that simply tests for task termination; this is not the case, and the main program may loop indefinitely (lines 74, 85, 86, 96, and 58, respectively).
- i. CD2B15C and CD7205C: These tests expect that a 'STORAGE SIZE length clause provides precise control over the number of designated objects in a collection; the Ada standard 13.2:15 allows that such control must not be expected.
- j. CD2D11B: This test gives a SMALL representation clause for a derived fixed-point type (at line 30) that defines a set of model numbers that are not necessarily represented in the parent type; by Commentary AI-00099, all model numbers of a derived fixed-point type must be representable values of the parent type.
- k. CD5007B: This test wrongly expects an implicitly declared subprogram to be at the address that is specified for an unrelated subprogram (line 303).
- l. ED7004B, ED7005C..D, and ED7006C..D (5 tests): These tests check various aspects of the use of the three SYSTEM pragmas; the AVO withdraws these tests as being inappropriate for validation.
- m. CD7105A: This test requires that successive calls to CALENDAR.CLOCK change by at least SYSTEM.TICK; however, by Commentary AI-00201, it is only the expected frequency of change that must be at least SYSTEM.TICK--particular instances of change may be less (line 29).
- n. CD7203B and CD7204B: These tests use the 'SIZE length clause and attribute, whose interpretation is considered problematic by the WG9 ARG.
- o. CD7205D: This test checks an invalid test objective: it treats the specification of storage to be reserved for a task's activation as though it were like the specification of storage for a collection.

WITHDRAWN TESTS

- p. CE2107I: This test requires that objects of two similar scalar types be distinguished when read from a file--DATA_ERROR is expected to be raised by an attempt to read one object as of the other type. However, it is not clear exactly how the Ada standard 14.2.4:4 is to be interpreted; thus, this test objective is not considered valid (line 90).
- q. CE3111C: This test requires certain behavior, when two files are associated with the same external file, that is not required by the Ada standard.
- r. CE3301A: This test contains several calls to END_OF_LINE and END_OF_PAGE that have no parameter: these calls were intended to specify a file, not to refer to STANDARD_INPUT (lines 103, 107, 118, 132, and 136).
- s. CE3411B: This test requires that a text file's column number be set to COUNT'LAST in order to check that LAYOUT_ERROR is raised by a subsequent PUT operation. But the former operation will generally raise an exception due to a lack of available disk space, and the test would thus encumber validation testing.

APPENDIX E
COMPILER OPTIONS AS SUPPLIED BY IBM

Compiler: The IBM Development System for the Ada Language
 AIX/RT Follow-on, Version 1.1

ACVC Version: 1.10

Give a description of any compiler options or switch settings used in running the tests, which do not use the default setting.

The "tkada" command is used to invoke the AIX/RT follow-on compiler. The input to the compiler is a file containing one or more Ada compilation units. Compilation is controlled by the options selected.

The syntax of the tkada command is:

```
tkada <options> <file or unit-name>
```

where:

file	This is the name of the Ada source file. The compiler assumes the default extension of ".ada" if none is present
+ assembly{ = <name> }	Produces assembly code output. <name> defaults to the source file name. Note that the assembly file produced contains a concatenation of all assembly code produced, and thus may not be legal format for the assembler. This will occur most often in the case of main unit compilations, where the MAIN elaboration code and the main program code will be concatenated into the same file.
+ bind	Produce an executable from previously compiled code. The Ada name of the main unit must also be specified on the command line.
+ bind = <unit-name>	Bind the named unit as a main program, after compiling the source file also specified on the command line. The unit to be bound as main need not have been compiled in the named source file, although this is allowed.
-cg_debug	Specify that code generator debugging information should not be performed. This is the default setting.
-cg_optimize	Specify that the code generator may not perform optimizations that may interfere with debug tools.
+ copy	Copy the source file into the "src" subdirectory of the working sublibrary. This is the default. When + copy is used, the banner displayed by the compiler will show that the source file is the copy in the "src" subdirectory of the working sublibrary.
-copy	Do not copy the source file in to the "src" subdirectory of the working sublibrary.
+ cleanup	Reset the working sublibrary. Use this option to clear the working sublibrary after a previous compilation has been killed. Any other options specified in the same command are ignored.
+ include = <name>	Include the named object code or archive file in the linking of the main program. This option can be used to specify that non-standard object code libraries contain library routines needed by the Ada program. <name> may also specify an option to be sent to the AIX linker invoked to build the main program. This option is ignored if neither "+ main" nor "+ bind" was specified for the compilation. If more than one file or option is to be specified to the linker, use "+ include" once for each one.

+instr(= produce)	Indicate that the main program being built will produce an instrumentation output file reporting basic block profiling information when run. An AIX script file with the form <out-name>.ins will be generated which will reinvoke the compiler to take advantage of profiling information. <out-name> is selected with the +out option and defaults to MAIN.
+level_ < num >	Select the optimization level to be used. < num > may be 0,1, or 2. +level_0 performs the fewest optimizations and provides the fastest compilation speed; +level_2 performs the most optimizations and requires the most compilation time. This option may be abbreviated by omitting characters between "+" and < num >.
+library = < lib >	Specify the Ada library list name (defaults to liblst.alb). N.B. The library name "liblst.tmp" is reserved and should not be used.
+listing	Generate Source Listings.
-link	Quit without linking an executable. Valid only with "+main" or "+bind". The linker script (with the name < out-name > .lnk can be executed separately to invoke the linkage step.
+main	Compile the named source file, and bind the last unit in the file as a main program.
+monitor	Same as +verbose
-monitor	Same as -verbose
+verbose	Print progress messages during compilation.
-verbose	Operate silently, without progress messages.
+object	Produce object code output (as opposed to assembly code).
+out = < output >	Specify the name to be given to the executable output and informational files. The default is MAIN. < output > should not be more than 6 characters long.
+optimize	Turn on the global optimizer.
-optimize	Turn off the global optimizer.
+pre_process = < option >	Enables compilation of source code lines identified by --(< option >) comments.
-pre_process = < option >	Disables compilation of source code lines identified by --(< option >) comments.
+save	Do not delete the < out > .lnk file used to link the main program. This option is valid only with "+main" or "+bind". The < out > .lnk file may be re-invoked separately to re-link the main program without invoking the compiler.
+virtual = < num >	Specify the number of virtual pages used by the virtual space manager. A larger number allows faster compilation speed, but demands more memory from the host system. In general, decrease "+virt" if the compiler runs out of memory, and increase "+virt" for systems with an abundance of memory (page space). The default is 3000.

Each option can be abbreviated to the first three characters.